Applikation för hantering av arbetsorder

Daniel Holmström

Examensarbete för Ingenjörs (YH)-examen

Utbildningsprogrammet för Informationsteknik

Raseborg 2011

YRKESHÖGSKOLAN
NOVIA

EXAMENSARBETE

Författare: Daniel Holmström
Utbildningsprogram och ort: Informationsteknik, Raseborg
Handledare: Klaus Hansen

Titel: Applikation för hantering av arbetsorder

Datum 27.03.2011 Sidantal 39 Bilagor

Sammanfattning

De flesta företag använder sig av så kallade arbetsorder för uppföljning av olika arbeten som utförs eller har utförts, på Ekenäs Energi Ab har denna uppföljning hittills gjorts med penna och papper. I det här arbetet beskrivs hur ett system för hantering av dessa arbetsorder utvecklats. Systemet gör det möjligt för personalen att via sina arbetsdatorer lägga till och följa upp dessa utförda uppdrag. Systemet byggs upp som ett Windows-baserat program för att huvudsakligen användas på arbetsdatorer inom Ekenäs Energi.

Som utvecklingsverktyg för programmet har Microsoft Visual C# Express Edition använts som programmeringsspråk och Microsoft SQL Server 2008 för databashantering. Syftet var att med hjälp av dessa verktyg utveckla ett program med lättförståeligt och smidigt användargränssnitt, själva programmet utvecklades för bruk på datorer med operativsystemet Windows XP.

Efter en tre månaders testperiod och vissa modifikationer på utseende och funktionalitet togs den slutgiltiga versionen i bruk av Ekenäs Energi i januari 2011. Arbetsledningen vid företaget var mycket nöjd med programmet som förmodligen kommer att både underlätta och effektivisera företagets framtida arbete.

Språk: svenska Nyckelord: arbetsorder, programmering, C#, ERP, affärssystem

---

YRKESHÖGSKOLAN
NOVIA

BACHELOR'S THESIS

Author: Daniel Holmström
Degree Programme: Information Technology
Supervisors: Klaus Hansen

Title: Application for Managing Work Orders/ Applikation för hantering av arbetsorder

---

Date 27 March 2011 Number of pages 39 Appendices

Summary

Most corporations today use so called work orders to monitor different jobs that are being performed or have been completed. The company Ekenäs Energi Ab has been using pen and paper to monitor their jobs thus far. This thesis deals with how a new system for managing these work orders has been developed making it possible for the employees at Ekenäs Energi to initiate and monitor these jobs through their computers. This system will be designed as a Windows-based program intended for use on work computers at Ekenäs Energi.

The tools that have been used in this project are: Microsoft Visual C# Express Edition as the programming language and Microsoft SQL Server 2008 as the database handler. The purpose is to use these tools to develop a simple and smooth user interface. The application itself is being developed for use on Windows XP-based computers.

After a three-month test period and some modifications of the interface and functionality of the application, the final version was commissioned in January 2011. The company's work administration was very pleased with the program as a whole and it will most likely make their future work both more efficient and easier.

Language: Swedish Key words: work order, programming, C#, ERP

Innehållsförteckning

1 Inledning

1.1 Bakgrund och syfte

Detta examensarbete görs som ett beställningsarbete åt Ekenäs Energi Ab . Syftet är att utveckla ett hanteringssystem för arbetsorder i form av ett datorprogram. Meningen är att personalen vid Ekenäs Energi (förkortas i fortsättningen EE) via sina arbetsdatorer skall kunna initiera, fylla i, följa upp och skriva ut arbetsorder som kommer att finnas lagrade i en databas.

Inom EE omfattar personalens arbetsorder alla de el-reparations-, utbyggnads- och underhållsarbeten som utförs inom Raseborg. Dessa kan t.ex. vara underhåll av elstationer och utbyggnad av olika distributionsnät. Arbetsledningen i EE är de personer som ansvarar för att starta upp nya arbetsorder och övervaka dessa medan de finns en huvudansvarig montör (per arbete) som bär ansvaret för ifyllning av arbetets framskridande. Efter att arbetet slutförts skickas arbetsordern till ett materiallager som godkänner ordern innan den sedan skickas vidare till bokföringen.

Hittills har personalen på EE i sin verksamhet använt sig av penna och papper för att fylla i olika arbetsorder över utförda arbeten, arbetsledningen på EE ansåg dock att detta var ineffektivt och ville digitalisera sina arbetsorder. Meningen är att företaget så småningom skall övergå till ett kommersiellt program för både insättning av utförda arbetstimmar och arbetsorder, men detta system är ganska invecklat konstruerat och en hel del detaljer måste konfigureras innan programmet ens fungerar. P.g.a detta behövde EE ett mera användarvänligt program fram till dess att det nya systemet tas i bruk.

1.2 Företaget

Ekenäs Energi är ett kommunalt affärsverk för elproduktion i staden Raseborg, företaget inledde sin verksamhet år 1909 och deras huvudkontor finns för närvarande på Kungsgatan 14 i Ekenäs. De anställdas antal uppgår till ca. 30, majoriteten arbetar med olika montörs- och reparationsarbeten. Det är dessa

personer som ansvarar för att fylla i arbetsorder över utförda arbeten. De av personalen som hör till arbetsledningen har till uppgift att starta upp och övervaka de arbeten som utförs.

1.3 Begränsningar

Tyngdpunkten i detta examensarbete kommer att vara på det grafiska användargränssnittet samt programkoden bakom detta. I arbetet framgår inte i detalj hur databasen är uppbyggd (förutom det ER-diagram som finns i Figur 7) och hur databastabellerna förhåller sig till EE:s befintliga system för uppföljning av arbetstimmar.

2 Kravspecifikation

För att kunna förverkliga detta projekt gjordes en plan upp för hur det färdiga programmet skulle se ut, vilka kriterier det måste uppfylla samt en rimlig tidpunkt då själva programmet kan tas ibruk i testningsyfte och hur länge denna period skall pågå.

2.1 Projektmål

Den slutliga produkten kommer att vara ett Windows-baserat program med vilket användare kan tillföra nya arbetsorder till en databas, öppna och fylla i dessa medan arbetet framskrider samt skriva ut papperskopior på dem.

2.2 Funktionella krav

- Inloggning med befintliga användarnamn och lösenord till en databasserver.
- Endast arbetsledningen skall kunna tillföra nya arbetsorder till databasen.
- Montörer skall endast kunna se de arbetsorder som de själva bär huvudansvararet för.

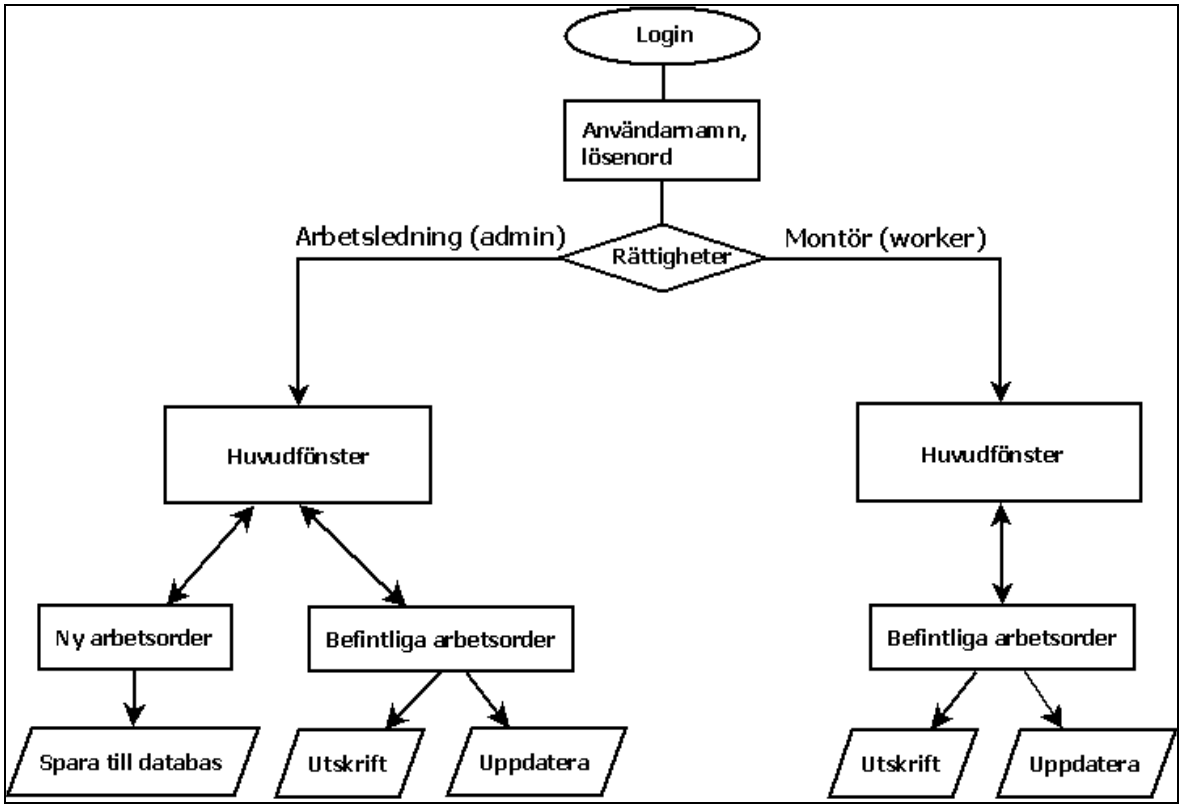
- Arbetsorder som tillförts till databasen skall ej kunna raderas genom detta program.

2.3 Flödesschema

För att kunna skapa en bättre överblick av hur programmet skulle fungera i praktiken ritade beställaren upp ett flödesschema för programmet (Figur 1). Här ser vi att programmet startar med ett inloggningsfönster (Login) som läser in användarnamn och lösenord varefter rättigheter som antingen arbetsledning (admin) eller montör (worker) tilldelas. Sedan öppnas fönstret för översikt och sökning av arbetsorder (Huvudfönster) oberoende av om användaren tillhör ”admin” eller ”worker”, däremot kommer vissa funktioner vara otillgängliga för användare i ”worker” kategorin.

För användare med admin-rättigheter skall alla funktioner i programmet att vara tillgängliga. Denna användarkategori kan även se alla befintliga arbetsorder i databasen oavsett vem som utför arbetet samt lägga till nya arbetsorder via fönstret för nya arbetsorder.

Om en användare hör till worker men inte admin, mao utför tekniska arbeten, men saknar administrativa rättigheter, kommer, förutom huvudfönstret, endast fönstret för befintliga arbetsorder att vara tillgängligt. Användare som hör till worker kommer dessutom endast att kunna se och uppdatera de arbetsorder som användaren själv ansvarar för.



Figur 1. Flödesschema som beställaren ritat upp

3 Utvecklingsverktyg & design

Själva applikationen byggs upp som ett Windows- baserat program med hjälp av programkompilatorn Microsoft Visual C# 2008 Express Edition, som databashanterare används Microsoft SQL Server 2008 (Microsoft förkortas i fortsättningen MS).

3.1 MS Visual C# 2008 Express Edition

Som programkompilator används gratisversionen MS Visual C# 2008 Express Edition, C# (uttalas C sharp) som är ett objektorienterat programmeringsspråk vars syntax till stor del liknar föregångaren C++ och Java. Mannen som leder utvecklingen av språket för MS är en dansk mjukvaruingenjör vid namn Anders Hejlsberg. Syftet bakom utvecklingen av C# var att ta fram ett programmeringsspråk som (bl.a.): var

modernt, var enkelt att förstå och kunde upptäcka samt förhindra användningen av oinitierade variabler.

Inspirationen till namnet C# kommer från musikens värld där ”sharp”, mao en höjning av stamtonen, betecknas med just korsförtecknet #, denna symbol används även i andra programmeringsspråk, t.ex. J#.

Det som är noterbart med C# är avsaknadet av globala variabler och funktioner, dock är det möjligt att ersätta dessa med statiska publika varibler eller funktioner för att uppnå samma sak. Dessutom har C# förutom try-catch -satsen, också try-finally med vilken programmerare kan specificera att en viss kod alltid exekveras inom finally oberoende av utgången för try. Typsäkerhet är en annan av C# -finesser, detta innebär att endast de konverteringar som klassas som ”säkra” kommer utföras, t.ex. förlänging av integrer (heltal), denna kontroll utförs då själva programmet kompileras, dvs byggs upp till en exekverbar fil. (C# Programming language, u.å.b.)

3.2 MS SQL Server 2008

EE använder från tidigare databashanteraren MS SQL Server 2008. I själva databasen lagras data i olika tabeller, varje tabell kan i sin tur innehålla flera kolumner. Här finns stöd för bl.a. datatyperna: integer, float, varchar, bit och date.
(Microsoft SQL Server, u.å.l.)

MS SQL Server använder som frågespråk T –SQL (Transact – Structured Query Language), en variant av standard SQL (Structured Query Language) som tillåter användning av b.la. matematiska beräkningar i förfrågningarna samt möjligheten att kunna koppla samman tabeller med JOIN -uttrycket även då man uppdaterar (UPDATE) samt raderar (DELETE) information från databasen.
(Transact –SQL, u.å.m.)

3.3 Extern design

Det första skedet i programutvecklingen var att få fram ett användargränsnitt som var enkelt att förstå men samtidigt praktiskt att använda. En stor del av den externa

designen är inspirerad av EE:s befintliga program för uppföljning av utförda arbetstimmar. Detta för att skapa en mera bekant arbetsmiljö för användarna jämfört med ett gränssnitt som ser radikalt annorlunda och främmande ut.

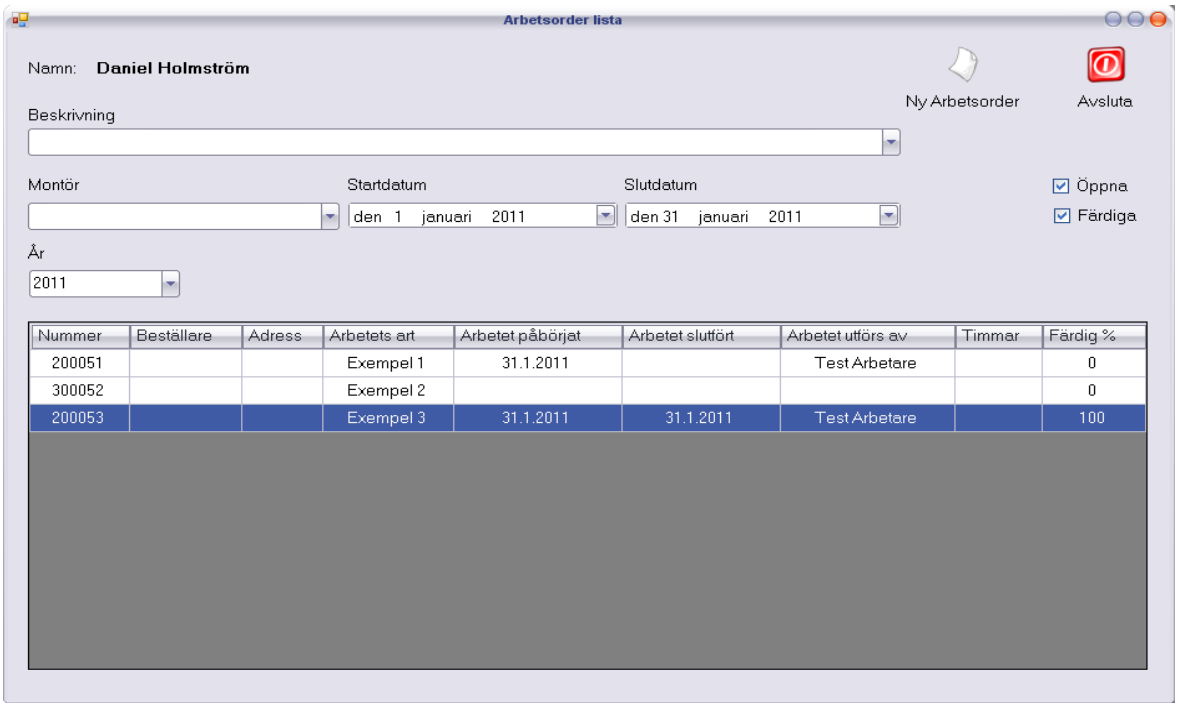
Det grafiska användargränsnittet presenteras nedan i kapitel 3.3.1 och programkoden som ligger till grund för denna senare i kapitel 3.4.

3.3.1 Användargränssnitt

- ett inloggningsfönster där användaren anger sitt användarnamn och lösenord (Figur 2).

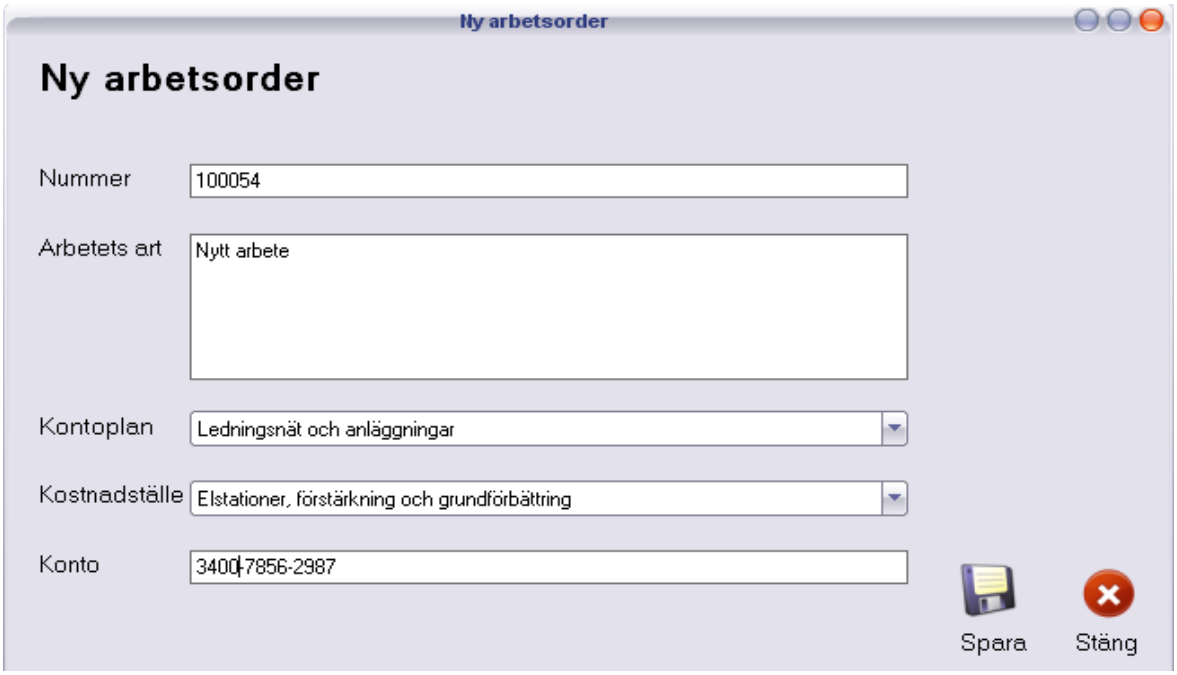
Figur 2. Inloggningsfönstret (FormLogin)

- ett huvudfönster med en vy över befintliga arbetsorder i mitten, rullgardinsmenyer för val av olika sökkriterier och knappar för skapande av nya arbetsorder och stängning av själva programmet (Figur 3).



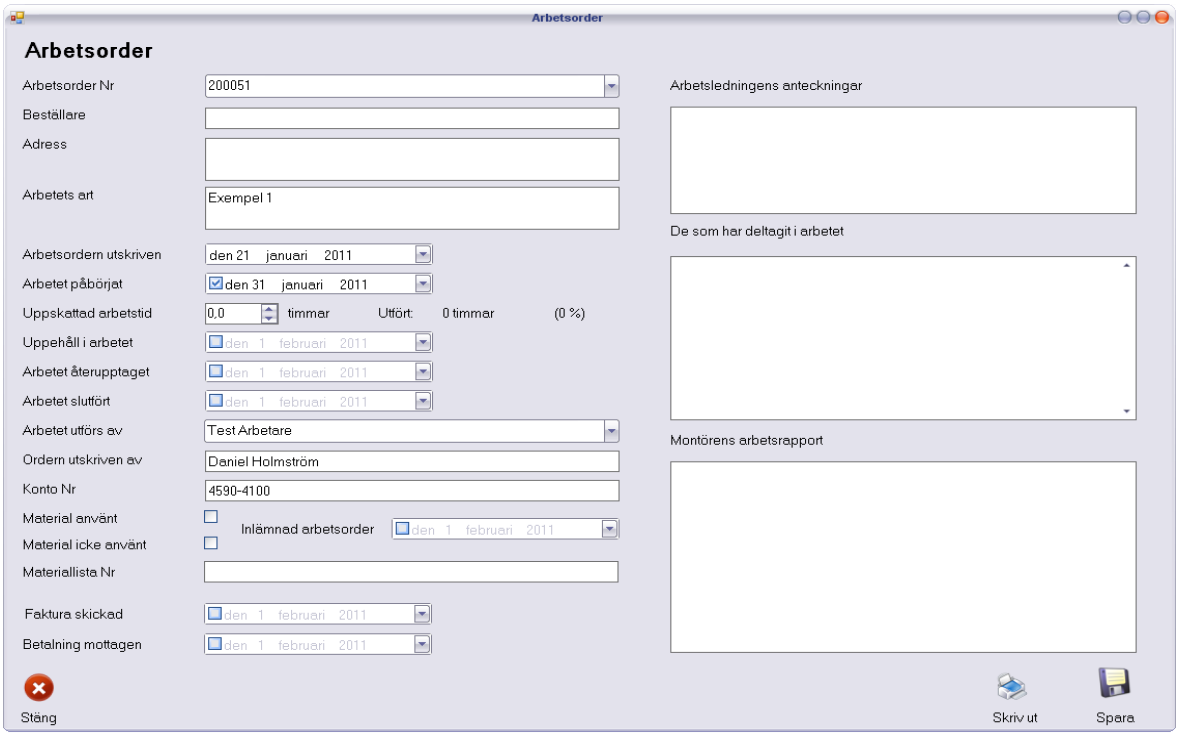
Figur 3. Huvudfönstret (FormMain)

- ett fönster för skapande av nya arbetsorder (Figur 4).



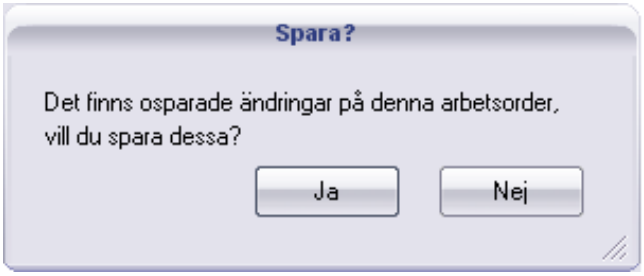
Figur 4. Tillförande av nya arbetsorder till databasen (FormNew)

- ett skilt fönster för befintliga arbetsorder i databasen där det finns möjlighet att redigera och skriva ut dessa (Figur 5).



Figur 5. Överblick av specifik arbetsorder (FormOrder)

- en dialogruta som används för att spara ändringar i arbetsorder till databasen (Figur 6).



Figur 6. Dialogruta för sparande av ändringar (FormSaveBox)

3.4 Intern design

Programmet kommer att bestå av fyra st. formulär för de olika fönster som finns samt ett formulär som kommer att fungera som en dialogruta.

3.4.1 Översikt av databasen

EE har sedan tidigare en databas som innehåller bl.a. information om utförda arbetstimmar, använda fordon och olika tilläggsprocenter. För att denna databas också skall kunna omfatta arbetsorder måste ett antal tabeller läggas till, dessa är: ee_jobOrder_list, ee_account_pre och ee_account_post (Figur 7). Från dessa samt från ee_workers används samtliga kolumner, från de övriga tabellerna används endast vissa fält.

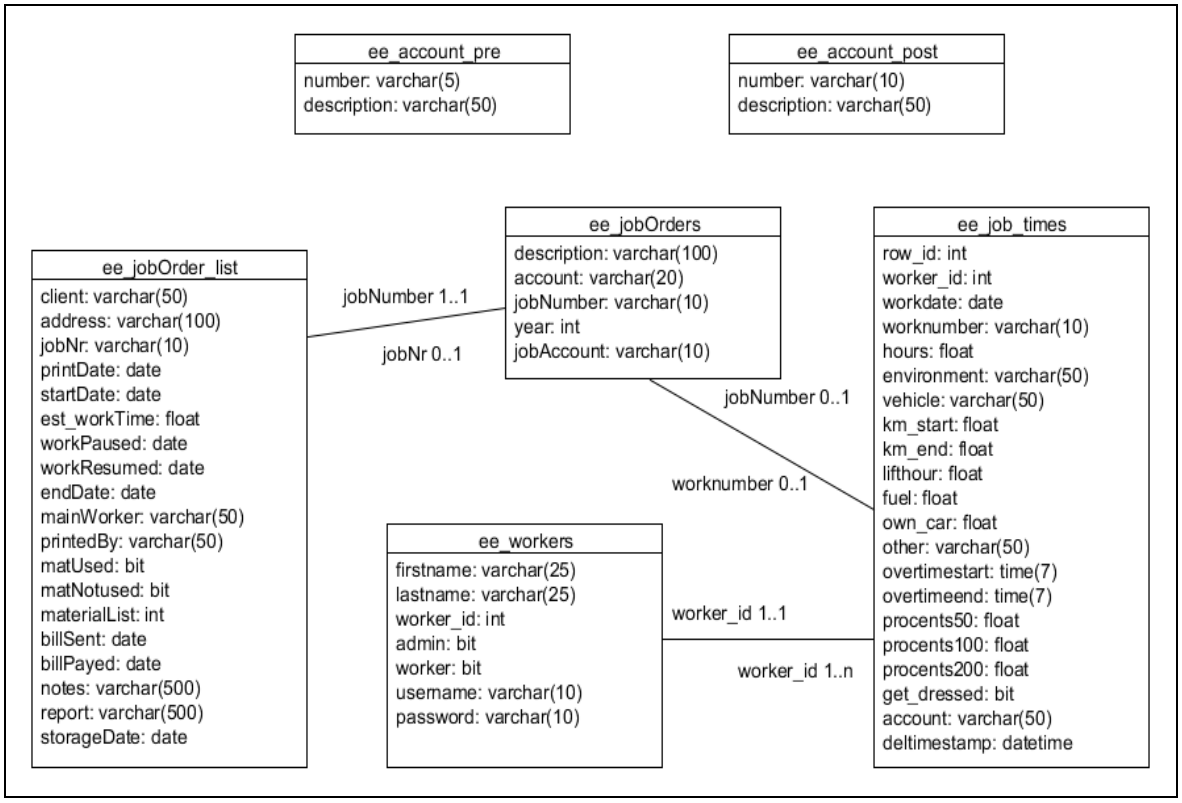
Tabellen ee_jobOrder_list innehåller det mesta av informationen i en arbetsorder, de kolumner som är markerade som varchar() är textfält där en valfri kombination av text, symboler eller siffror kan sättas in (förutom citattecken). Kolumner med ”date” innebär helt enkelt ett fält för datuminsättning, ”float” är ett fält för decimaltal och för de två ikryssningsrutorna på FormOrder används ”bit” vilken antigen är ”null” (inget värde) eller 1. Nyckelkolumnen i den här tabellen är jobNr, mao arbetsordernummer.

Kontonummer finns lagrade i tabellerna ee_account_pre och ee_account_post, det fullständiga kontonumret för vissa arbetsorder fås genom att kombinera siffror från dessa två tabeller.

I tabellen ee_jobOrders kopplas jobNr (arbetsorderns nummer) samman med jobNumber och här finns även den viktigaste funktionen, nämligen att hålla reda på från vilket år en viss arbetsorder är insatt. Dessutom finns här beskrivning och kontonummer (account), jobAccount används inte av detta program.

Från tabellen ee_job_times används endast kolumnerna worknumber, procents50, procents100, procents200 samt deltimestamp. Worknumber hör ihop med jobNr och jobNumber medan procents 50-200 har med utförda arbetstimmar att göra och deltimestamp innehåller datum när arbeten är inlämnade.

Fullständiga namn, användarnamn , lösenord samt tillhörighet (admin eller worker) finns i tabellen ee_workers. Här får dessutom alla insatta användare ett unikt användarnummer (worker_id) för att lättare kunna koppla samman arbeten med montörer som utför dem.



Figur 7. ER –diagram över de använda databastabellerna

3.4.2 Inloggningsfönstret (FormLogin)

När programmet startas öppnas först ett inloggningsfönster där användaren måste ange användarnamn och lösenord för att kunna komma in till huvudfönstret (se Figur 2). I inloggningsmomentet jämför programmet den inmatade informationen med den som finns i databasen, om ett felaktigt användarnamn eller lösenord uppges så öppnas en dialogruta som uppmanar att kolla den inmatade informationen. Om informationen stämmer tilldelas antingen administrativa (admin) rättigheter eller begränsade (worker) rättigheter beroende av vilken användargrupp personen hör till, efter detta öppnas huvudfönstret av programmet.

3.4.3 Inloggninsprocedur

Koden i Kod 1 visar hur programmet hanterar kontakten till databasen då användaren loggar in. Först bildas en SQL-sträng där förnamn, efternamn samt rättigheter (admin eller worker) hämtas från databasen och de inmatade värdena i

textBoxUser (användarnamn) samt textBoxPass (lösenord) jämförs med de värden som finns i databasen. För att kunna öppna kontakten till databasservern anropas en fördefinierad sträng (strCon) från ett annat formulär, FormMain (se Kod 2), och ett förmedlingssätt till databasen (dataAdapter). Adaptern använder sedan SQL-strängen och skickar den vidare till databasen, denna princip med en SqlDataAdapter används varje gång en förfrågning skickas till databasen.

Om användarnamn och lösenord stämmer överens med de värden som finns i databasen kommer en ny rad att dyka upp i tabellen och en räkning (count) kommer att ge värdet ”ett” varvid en variabel, userName, definieras med användarens förnamn och efternamn (firstname, lastname). Efter detta stängs inloggningsfönstret och huvudfönstret för programmet (FormMain) öppnas. Om användaren inte finns i databasen eller lösenordet som givits är felaktigt kommer räknaren att returnera värdet ”noll” och en meddelanderuta dyker upp där användaren ombeds kontrollera de inmatade uppgifterna. Fältet för inmatning av lösenord töms också i detta skede.

Kod 1. Inloggningsmetod

```
private void button1_Click(object sender, EventArgs e)
{

    string strSQL = "SELECT firstname, lastname, admin, worker "
                  + "FROM ee_workers WHERE username = '" + textBoxUser.Text + "' "
                  + "AND password = '" + textBoxPass.Text + "'";

    SqlDataAdapter dataAdapter = new SqlDataAdapter(strSQL, FormMain.strCon);

    DataTable table = new DataTable();

    dataAdapter.Fill(table);

    if (table.Rows.Count == 1)
    {
         FormMain.userName = table.Rows[0]["firstname"].ToString() + " " +
         table.Rows[0]["lastname"].ToString();
          ***
          this.DialogResult = DialogResult.OK;

    }
    else
    {
          MessageBox.Show("Kontrollera användarnamn och lösenord!");
          textBoxPass.Text = "";
    }
  }
```

3.4.4 Huvudfönstret (FormMain)

I huvudfönstret (se Figur 3) får man en överblick av alla tillgängliga arbetsorder i databasen från ett rutfält för visning av data, DataGridView (förkortas i fortsättningen DGV), samt olika sökkriterier som: ”beskrivning”, ”montör”, ”datum”, ”år” och ”öppna” samt ”färdiga” för att enkelt kunna filtrera innehållet i denna. Om användaren inte har administrativa rättigheter kommer rullgardinsmenyn för montör inte att vara synlig eftersom en sådan användare endast kan se de arbetsorder som han/ hon själv är ansvarig montör för.

När det gäller sökning av tillgängliga arbetsorder kommer alltid ”år” samt ”start/ slut datum” att vara med och kan aldrig väljas bort medan ”beskrivning”, ”montör” och ”öppna / färdiga” kriterierna är valfria. Alla sökningar kommer dock alltid att beakta alla andra valda kriterier, t.ex. kommer montör- och beskrivningsmenyerna inte visa pågående arbetsorder om ikryssningsrutan för ”Öppna” inte är markerad.

3.4.5 Öppning av huvudfönstret

I Kod 2 visas den publika SQL-sträng som sköter kontakten till databsen (strCon), denna innehåller IP-adressen till databasservern samt från vilken katalog förfrågningen skall tas och ett användarnamn och lösenord till servern. Här finns även två andra mycket viktiga publika variabler,nämligen userName och year, dessa innehåller information om vilken användare som är inloggad resp. från vilket år arbetsorder skall visas. Här definieras också två boolska värden, ”admin” och ”worker”, med vilka rättigheter till den inloggade användaren anges. Den publika strängen strCon återanvänds i programmet varje gång kontakt med databasen sker.

Under FormLoad, dvs metoden som körs då fönstret öppnas, finns en villkorsats (IF-sats) där FormLogin initieras och FormMain förblir osynligt tills inloggningen lyckats (Dialogresult.OK) då detta fönster görs synligt och alla startvärden läggs ut. Metoden comboYear väljer ut alla befintliga år som finns insatta i databastabellen ee_jobOrder och sätter in dessa i rullgardinsmenyn ”År”. För att kunna hämta det nuvarande året används DateTime.Now.Year, detta läggs in som startvärde i val av år.

Användarens fullständiga namn fås från userName och läggs in som en etikett (lableName) i fönstrets övre vänstra kant. Startdatumet för sökningar skall alltid börja med 1 Januari i det aktuella året, detta uppnås genom: DateTime.Now.Year, 1, 1.

Kod 2. Metoden som exekveras då FormMain öppnas

```
public static string strCon = "Data Source=192.168.***.**\SQLEXPRESS;Initial
Catalog=energiDB;User ID=sa;Password=*********", userName, year;

private void FormMain_Load(object sender, EventArgs e)
{
    FormLogin frm = new FormLogin();
    if (frm.ShowDialog(this) == DialogResult.OK)
    {
        comboYear();
        string yearbox = System.DateTime.Now.Year.ToString();
        comboBoxYear.Text = yearbox;
        year = comboBoxYear.Text;
        labelName.Text = userName;
        dateTimePickerStart.Value = new DateTime(DateTime.Now.Year, 1, 1);
    }
    else
    {
        this.Close();

    }
```

Om användaren tillhör kategorin worker men inte admin kommer möjligheterna att tillföra nya arbetsorder samt att filtrera sökresultat enligt montör att vara otillgängliga. Detta åstadkoms genom metoden FormMain_Shown, alltså metoden som exekveras då fönstret blir synligt, där dessa funktioner görs osynliga med: Visible = false (Kod 3).

Kod 3. Metoden för undangömmning av vissa funktioner

```
private void FormMain_Shown(object sender, EventArgs e)
{

    if (!admin && worker)
    {
        pictureBoxNew.Visible = false;
        comboBoxFitter.Visible = false;
        labelNew.Visible = false;
        labelFitter.Visible = false;

    }
 }
```

3.4.6 Sökning av data i huvudfönstret

Då användaren utför en sökning i databasen tillkallas metoden GetData (Kod 4). Metoden är fristående, mao inte bunden till en viss händelse som t.ex. en knapptryckning utan kan tillkallas av vilken annan händelse som helst. (ASPNet, u.å.a.)

Här används återigen strCon för att kontakta databasen samt stringbuilder (openStr) med vilken det är möjligt att koppla ihop en längre teckensträng av olika små delar beroende av vilka sökkriterier som matats in. Med Append-kommandot kan information tillföras direkt till strängen utan att skriva över det som redan finns där. SET DATEFORMAT DMY används alltid i början av en SQL-sträng och konverterar datumformatet till dag/månad/år, detta är viktigt med tanke på att sökresultaten kommer att bli felaktiga om inte samma format används både av programmet och databasen.

Då användaren inte tillhör admin (!admin && worker) kommer alla resultat från databasen endast vara sådana där användaren bär huvudansvaret för den specifika arbetsordern (jl.mainWorker = ’” + userName + ”’). Om varken fälten för beskrivning eller montör innehåller någon text (comboBoxNumber.Text == ”” && comboBoxFitter.Text == ””) och rutan för Öppna arbetsorder är ikryssad men inte rutan för Färdiga (checkBoxOpen.Checked && !checkBoxDone.Checked) kommer jl.endDate IS NULL att tilläggas i strängen, mao att slutdatum inte får finnas insatt i de resultat som returneras. Till slut används GROUP BY för att gruppera de olika sökkriterierna i kolumner då vi bl.a. använt en räkneoperation (SUM) samt ORDER BY job som ser till att de arbetsorder som visas i DGV är kronologiskt sorterade enligt de tre sista siffrorna. (Microsoft, u.å.i.)

Kod 4. Metoden för hämtning av data från databasen

```
public void GetData()
{
    try
    {

   openStr.Append("SET DATEFORMAT DMY
   +"SELECT jo.jobNumber, jo.description, jl.client, jl.address, SUM(jt.hours) as total "

   ***

     if (!admin && worker)
     {
        openStr.Append("AND jl.mainWorker = '" + userName + "' ");
     }

     if (comboBoxNumber.Text == "" && comboBoxFitter.Text == "")
     {
        if (checkBoxOpen.Checked && !checkBoxDone.Checked)
        {
           openStr.Append("AND jl.endDate IS NULL ");
        }

        ***

openStr.Append("GROUP BY jo.jobNumber, jo.description, jl.client, jl.address, jl.startDate,
             + "jl.endDate, jl.mainWorker, jl.est_workTime "
             + "ORDER BY job");
}
```

För att kunna filtrera sökresultaten baserat på om en arbetsorder pågår (Öppna) eller om arbetet är slutfört (Färdiga) används två stycken ikryssningsrutor, sk. checkboxes (Kod 5). Då det inte skall vara möjligt att ha båda dessa oikryssade används en IF-sats som kollar om båda dessa är oikryssade (!checkBox.Checked), om så är fallet kommer den rutan som användaren inte klickat på att bli ikryssad (checkBox.Checked = true).

Efter detta tillkallas metoden GetData() för att uppdatera sökresultaten.

Kod 5. Metoden för ändringar i ikryssningsrutorna

```
private void checkBoxOpen_CheckedChanged(object sender, EventArgs e)
{

    if (!checkBoxDone.Checked && !checkBoxOpen.Checked)
    {
       checkBoxDone.Checked = true;
    }

    GetData();
 }
```

När någon av rullgardinsmenyerna öppnas tillkallas en metod som fyller i sökresultaten i den menyn. I Kod 6 finns metoden bakom ”Beskrivning” –menyn, här används återigen stringbuilder för att få ihop en SQL-sträng av de olika kriterier som sökningen skall innehålla. Då resultatet returnerats används en WHILE-sats som lägger ut värdena till menyn (comboBoxNumber.Items.Add), denna princip gäller för alla rullgardinsmenyer i detta program. (Microsoft, u.å.k.)

Kod 6. Metod för tillförande av data till rullgardinsmeny

```
private void comboBoxNumber_DropDown(object sender, EventArgs e)
{
   ***

   try
   {
      ***

         nrStr.Append("SET DATEFORMAT DMY "
       + "SELECT DISTINCT jo.jobNumber, REPLACE(jo.description, '\r\n', ' '), jo.account,

        ***

      if (checkBoxOpen.Checked && !checkBoxDone.Checked)
      {
          nrStr.Append("AND jl.endDate IS NULL ");
      }

      ***

      nrStr.Append("ORDER BY job");

      ***
      while (reader.Read())
      {

         comboBoxNumber.Items.Add(String.Format("{0} {1} {2}", reader[0], reader[1],
         reader[2]));

      }
     }
```

3.4.7 Öppnande av befintliga arbetsorder

När användaren vill öppna en befintlig arbetsorder klickar denne på arbetsorderns nummer i listan över tillgängliga arbetsorder i DGV, varvid den valda arbetsordern öppnas i formuläret FormOrder (Kod 7). Noterbart är att det inte går att öppna en arbetsorder på något annat sätt än att klicka på nummer –fältet, i övriga fält kommer kolumnindexet vara större än noll (CurrentCell.ColumnIndex != 0) vilket innebär att ingenting sker då dessa klickas (CurrentCell.Selected = false). (Eggheadcafe, u.å.g.)

För att kunna lägga in de rätta värdena från en viss arbetsorder i fönstret FormOrder måste vi överföra numret som klickas till FormOrder, detta genom att först att hämta värdet i det fält som klickats och konvertera det till formatet string med: CurrentCell.Value.ToString(). Sedan läggs detta värde direkt in i textfältet för Arbetsorderns nummer i FormOrder. Då ett värde överförs från ett formulär till ett annat måste dessutom egenskaperna för det fält man vill skriva till vara publikt, annars kommer överföringen misslyckas. (Microsoft, u.å.j.)

Även här tillämpas principen att samma sida inte kan öppnas igen förrän den gamla har stängts med bool IsOpen = false / true (se Kod 8).

Kod 7. Metoden för öppnande av enskilda arbetsorder

```
private void dbGridView_CellClick(object sender, DataGridViewCellEventArgs e)
{

    FormOrder formOrder = new FormOrder();

    if (this.dbGridView.CurrentCell.ColumnIndex != 0)
    {
        this.dbGridView.CurrentCell.Selected = false;
    }

    else
    {

    bool IsOpen = false;

        foreach (Form f in Application.OpenForms)
        {
            if (f.Text == "Arbetsorder")
                {
                    IsOpen = true;
                    f.Focus();
                    break;
                }

            }

            if (IsOpen == false)
            {
                string nummer = dbGridView.CurrentCell.Value.ToString();
                formOrder.Show();
                formOrder.comboBoxDropDown();
                formOrder.comboWorker();
                formOrder.comboBoxNr.Text = nummer;
                formOrder.ComboValueChanged();
            }

        }
    }
```

3.4.8 Fönster för skapande av nya arbetsorder (FormNew)

Detta fönster är ämnat för att användare som hör till arbetsledningen på ett enkelt sätt skall kunna tillföra nya arbetsorder till databasen (se Figur 4). Det första fältet (Arbetsnummer) är delvis ifyllt från början eftersom det finns en metod som automatiskt söker upp det högsta förekommande numret baserat på de tre sista siffrorna i arbetsordern och adderar detta med ett, sedan läggs talet ut i fältet för arbetsnummer. Meningen är att användaren själv skall fylla i de tre första siffrorna samt en beskrivning av arbetet. Kontonummer fås antingen från att välja en kontoplan kombinerat med ett kostnadsställe om den första siffran i arbetsordern är 1, för siffrorna 2 och 3 finns speciella kontonummer som automatiskt läggs ut i kontonummerfältet.

Som en säkerhetsmekanism när man klickar på ”spara” kollar programmet först att det inte förekommer en arbetsorder med samma nummer i databasen. Efter detta sparas den inmatade informationen och programmet tömmer alla fält samt beräknar och lägger ut följande lediga arbetsnummer så att användaren direkt skall kunna lägga in en arbetsorder till om denne så önskar. Om numret man skrivit in redan förekommer i databasen meddelar programmet om detta och föreslår på nytt följande lediga nummer i Arbetsnummer fältet, alla andra fält förblir orörda.Till sist utförs även en kontroll av det inmatade kontonumret, så att användaren inte skall kunna spara felaktiga kontonummer till databasen.

För att förhindra att ett systemfel uppstår på grund av att användaren försökt spara ett arbetsordernummer som är kortare än sex siffror finns även en kontroll som visar en felmeddelanderuta om arbetsorderns nummer är kortare än sex siffror då användaren klickar på ”Spara” -knappen.

3.4.9 Öppnande av fönstret för inmatning av nya arbetsorder

Kod 8 visar koden bakom knappen ”Ny arbetsorder”. Om man redan har öppnat FormNew, mao formuläret för ifyllning av nya arbetsorder går det inte att öppna samma sida ännu en gång förrän användaren stängt det nuvarande fönstret. Detta görs genom att deklarera en boolsk variabel (IsOpen) och tilldela dess värde som

false, så länge IsOpen har detta värde är det möjligt att öppna formuläret. Då FormNew öppnas tilldelas denna variabel värdet true och om användaren i detta skede försöker gå tillbaka till huvudfönstret kommer detta att misslyckas, då den nuvarande sidan, f, har försatts med f.Focus(). Samma princip används även för FormOrder (formuläret som visar hela arbetsordern).

Kod 8. Metoden för "Ny arbetsorder" knappen

```
private void pictureBoxNew_Click(object sender, EventArgs e)
{

    bool IsOpen = false;

    foreach (Form f in Application.OpenForms)
    {
       if (f.Text == "Ny arbetsorder")
          {
             IsOpen = true;
             f.Focus();
             break;
          }

    }

    if (IsOpen == false)
    {
        FormNew formNew = new FormNew();
        formNew.Show();
    }

}
```

Då FormNew visas (Kod 9) skickas en förfrågan till databasen om vilket det högsta numret är (baserat på de tre sista siffrorna) från arbetsorder tillhörande det aktuella året med SELECT max(cast(right(convert (char(6), jobNumber) ,2) AS int)). För att kunna kontrollera om strängen som returnerats är tom används: string.IsNullOrEmpty() men före detta måste den koverteras till typen string med: String.Format("{0}", rdr[0]) . (Dotnetpearls, u.å.f.)

Om en tom sträng returneras betyder detta att inga arbetsorder är insatta under det aktuella året och programmet kommer att föreslå 100 i nummerfältet (textBoxNr.Text). Om en eller flera arbetsorder finns i databasen kommer de tre sista siffrorna i arbetsordern med högsta nummer att returneras, varvid programmet adderar detta med ett och föreslår detta nya nummer. En arbetsorder har alltid ett

sexsiffrigt nummer, de tre första skall användaren själv mata in, slutsiffrorna 001 – 099 är reserverade för speciella arbetsorder och dessa föreslås aldrig av programmet.

Kod 9. Metoden som exekveras då FormNew öppnas

```
private void FormNew_Shown(object sender, EventArgs e)
{

     try
     {
            ("SELECT max(cast(right(convert(char(6),jobNumber),3)as  int))+1 "
          + "FROM ee_jobOrders "
          + "WHERE year = year(GetDate())", conn);

               rdr = cmd.ExecuteReader();
               while (rdr.Read())
               {
                    string nummer = String.Format("{0}", rdr[0]);
                    if (string.IsNullOrEmpty(nummer) == true)
                    {
                        textBoxNr.Text = "100";
                    }
                    else
                    {
                        textBoxNr.Text = nummer;
                    }
               }
      }

      ***
}
```

3.4.10 Ifyllning av nya arbetsorder

En arbetsorder som läggs till i databasen kan endast innehåller siffror dvs inga bokstäver, punkter eller övriga specialtecken. Ett enkelt sätt att förhindra användare från att göra dessa misstag är att använda sig av e.Handled då någon tangenttryckning sker (Kod 10). En const char Delete som (char)8, mao alla siffror från noll till nio, deklareras sedan används e.Handled för att radera tangettryckningar som inte motsvarar dessa.

Kod 10. Raderar alla tangenttryckningar förutom siffror

```
private void textBoxNr_KeyPress(object sender, KeyPressEventArgs e)
{
    const char Delete = (char)8;
    e.Handled = !Char.IsDigit(e.KeyChar) && e.KeyChar != Delete;
}
```

Då användaren går in i textfältet för kontonummer kontrolleras vilken siffra det inmatade arbetsordernumret börjar på (string tal = textBoxNr.Text.Substring(0, 1)). Först konverteras ”tal” till ett heltal, (Convert.ToInt32(tal)) om denna siffra sedan är två eller tre läggs något av de fördefinierade kontonumren ut i fältet.

Om arbetsordernumret börjar med ett och användaren har valt en kontoplan samt kostnadsställekombination från de två rullgardinsmenyerna kommer programmet att lägga till ett kontonummer baserat på dessa istället (Kod 11).

Kod 11. Metoden som exekveras då användaren går in i kontonummerfältet

```
private void textBoxAccount_Enter(object sender, EventArgs e)
{

   string tal = textBoxNr.Text.Substring(0, 1);

   if (Convert.ToInt32(tal) == 2)
   {
       textBoxAccount.Text = "4590-4100";
   }

   else if (Convert.ToInt32(tal) == 3)
   {
       textBoxAccount.Text = "4590-4200";
   }

   else if (comboBoxPre.Text != "" && comboBoxPost.Text != "" && Convert.ToInt32(tal) != 2
&& Convert.ToInt32(tal) != 3)
   {
       try
       {
           SqlCommand cmdPre = new SqlCommand
                               ("SELECT number "
                              + "FROM ee_account_pre "
                              + "WHERE description = '" + comboBoxPre.Text + "'", conn);

           rdrPre = cmdPre.ExecuteReader();
           while (rdrPre.Read())
           {
               textBoxAccount.AppendText(String.Format("{0}-", rdrPre[0]));
           }

           rdrPre.Close();

           ***
}
```

3.4.11 Tillförande av nya arbetsorder till databasen

I Kod 12 finns koden som exekveras då ”Spara” -knappen klickas, här skickas återigen en förfrågning om det största befintliga numret i databasen varefter siffrorna i fältet för arbetsorderns nummer måste uppfylla vissa krav för att programmet skall gå vidare. Om det inmatade numret är kortare än sex siffror (textBoxNr.Text.Length != 6) dyker genast en meddelanderuta upp som informerar om detta, en arbetsorders nummer skall alltid innehålla sex siffror. Efter detta jämförs de tre sista siffrorna från nummerfältet (textBoxNr.Text.Substring(textBoxNr.Text.Length – 3, 2) med de som nyligen hämtats från databasen (string compare). Om dessa är identiska går programmet vidare till metoden för kontroll av kontonummer (accountCheck), i annat fall meddelas användaren att numret är felaktigt.

Kod 12. Metoden för ”Spara” -knappen

```
private void pictureBoxSave_Click(object sender, EventArgs e)
{
     try
     {
        ***

         ("SELECTmax(cast(right(convert(char(6),jobNumber),3)as int))+1 "
        + "FROM ee_jobOrders "
        + "WHERE year = year(GetDate())", conn);

         rdr = cmd.ExecuteReader();
         while (rdr.Read())
         {
            if (textBoxNr.Text.Length != 6)
            {
                MessageBox.Show("Arbetsorderns nummer måste vara 6 siffror långt!");
            }

            else
            {
                string compare = String.Format("{0}", rdr[0]);
                string number = textBoxNr.Text.Substring(textBoxNr.Text.Length - 3, 3);
                string lastdigits = textBoxNr.Text.Substring(textBoxNr.Text.Length - 2, 2);


                if (string.IsNullOrEmpty(compare) && Convert.ToInt32(lastdigits) == 100)
                {
                   accountCheck();
                }
                else if (compare.Length == 3 && number == compare)
                {
                    accountCheck();
                }

                ***
            }
         }
      }

      catch(SqlException)
      {
          MessageBox.Show("Det gick inte att lägga till arbetsordern!");
      }
  }
```

Efter att programmet kontrollerat arbetsorderns nummer skall även det inmatade kontonumret granskas (Kod 13), detta för att förhindra eventuella felaktiga kontonummer som användare kan mata in av misstag. Genom att skicka en SELECT COUNT-sträng till databasen och hämta alla möjliga kontonummerkombinationer från ee_account_pre och ee_account_post samt de två fördefinierade kontonumren och sedan kontrollera om räknaren hittat ett matchande nummer till det inmatade numret (Convert.ToInt32(konto) != 1). Om kontonumret stämmer överens med någon av de sifferkombinationer som finns i databasen går programmet vidare till själva insättningen av arbetsordern (newSave).

Kod 13. Metoden för kontroll av kontonummer

```
private void accountCheck()
{

   try
   {
        ***

        ("SELECT COUNT(*)amount "
       + "WHERE '" + textBoxAccount.Text + "' IN "
       + "(SELECT pre.number+'-'+post.number "
       + "FROM ee_account_pre AS pre, ee_account_post AS post "
       + "UNION SELECT '****-****' "
       + "UNION SELECT '****-****')", conn);


        rdr = cmd.ExecuteReader();
        while (rdr.Read())
        {
            string konto = rdr[0].ToString();

            if (Convert.ToInt32(konto) != 1)
            {
                MessageBox.Show("Kontonumret är felaktigt, vänligen korrigera!");
            }
            else
            {
                newSave();
            }
         }
       }

       ***

 }
```

Metoden newSave() i Kod 14 består av en INSERT INTO-sträng där alla värden sätts in som null, förutom de inmatade värdena i formuläret samt vilken användare som lagt till arbetsordern (printedBy), dagens datum som kan hämtas med GetDate() och aktuellt år med year(GetDate()).

Kod 14. Metoden för tillförande av nya arbetsorder

```
public void newSave()
{

   try
   {
      ***

       ("INSERT INTO ee_jobOrders (jobNumber, description, account, year) "
      + "VALUES ('" + textBoxNr.Text + "', '" + textBoxDescription.Text.Replace("\"","

      ***
   }
   ***

 }
```

Efter att användaren lagt in en ny arbetsorder i databasen tillkallas reset() som stänger fönstret och sedan öppnar det på nytt (Kod 15), detta är ett enkelt sätt att återställa alla värden så att användaren direkt kan lägga till följande arbetsorder om denne så önskar.

Kod 15. Återställande av alla värden

```
public void reset()
{
    this.Close();
    FormNew formNew = new FormNew();
    formNew.Show();

}
```

3.4.12 Hantering av arbetsorder (FormOrder)

Genom att dubbelklicka på ett arbetsnummer i DGV öppnas ett fönster med den valda arbetsordern (se Figur 5). Här finns möjlighet att ändra och lägga till all information som ordern skall innehålla, om användaren har administrativa rättigheter kan alla fält, förutom själva arbetsnumret samt fältet för ”Deltagit i arbete” ändras. För de användare som ej har administrativa rättigheter kommer antalet fält som är redigerbara att vara betydligt färre. Det är även möjligt att skriva ut en kopia av aktuell arbetsorder eller uppdatera informationen i en arbetsorder genom att klicka på ”Skriv ut” -respektive ”Spara” -knapparna.

Då en arbetsorder vars slutdatum är insatt i databasen väljs, betraktas denna som slutförd och alla fält förutom datumväljarna ”Faktura skickad” och ”Betalning mottagen” är låsta och kan inte redigeras. Om användaren har administrativa rättigheter dyker det i detta fall upp en knapp med utseendet som ett halvt öppnat hänglås med texten ”Lås upp” bredvid ”Skriv ut” -knappen. Genom att klicka på denna knapp låses alla fält upp tillfälligt så att korrigeringar av eventuella fel skall vara möjliga, arbetsordern återgår till låst läge automatiskt nästa gång den väljs.

3.4.13 Öppnande av fönstret för hantering av befintliga arbetsorder

När sidan för hantering av enskilda arbetsorder (FormOrder) öppnas kommer vissa fält att vara låsta (<kontrollens namn>.Enabled = false) och går därmed inte att redigera om användaren saknar administrativa rättigheter (Kod 16).

Kod 16. Metoden som exekveras då FormOrder öppnas

```
private void FormOrder_Shown(object sender, EventArgs e)
{

      if (!admin && worker)
      {

          textBoxClient.Enabled = false;
          textBoxAddress.Enabled = false;
          textBoxType.Enabled = false;

          ***
       }

 }
```

För att kunna lägga ut alla värden från en viss arbetsorder till fönstret FormOrder behövs en metod som tillkallas när värdet i ”Arbetsnummer” -fältet ändras (Kod 17). Först byggs en SQL-sträng upp och skickas till databasen, här gäller också att om användaren inte är admin kommer endast arbetsorder som användaren är ansvarig för att visas.

För vanliga textfält konverteras resultaten direkt till en textsträng med: Convert.ToString() och kan sedan placeras ut i textrutan. Vad gäller datumfält kollas det först att SQL-strängen inte är tom med: !string.IsNullOrEmpty(), varefter Parse används för att få ut ett datum ur strängen, detta sköts av: DateTime.Parse(). (Dotnetpearls, u.å.e.)

De två ikryssningsrutorna ”Material använt” och ”Material icke använt” kollas på samma sätt som datum, men om de innehåller ett värde blir rutan automatiskt ikryssad eftersom de bara kan ha två lägen.

Kod 17. Metoden för hämtning av data till FormOrder

```
public void ComboValueChanged()
{
     try
     {

        showStr.Append("SET DATEFORMAT dmy "
                     + "SELECT jo.description, jo.account, jl.client,
                     ***

        if (!admin && worker)
        {
            showStr.Append("AND jl.mainWorker = '" + FormMain.userName + "' ");
        }

        ***

        showStr.Append("GROUP BY jo.description, jo.account, jl.client,
        ***

        while (sel.Read())
        {
             textBoxType.Text = Convert.ToString(sel[0]);

             string print = Convert.ToString(sel[4]);

             if (!string.IsNullOrEmpty(print))
             {
                 datePrinted.Value = DateTime.Parse(print);
             }

             if (!string.IsNullOrEmpty(used))
             {
                  checkBoxUsed.Checked = true;
             }
            ***

       finally
       {
           ***
       }
 }
```

3.4.14 Uppdatering av befintliga arbetsorder

För att förhindra användare från att stänga en arbetsorder utan att spara de ändringar som gjorts visar Kod 18 en enkel säkerhetmekanism som jämför de värden som finns inmatade i fönstret med de som är sparade i databasen. En vanlig SELECT-sats väljer först ut alla värden från aktuella arbetsordern som finns insatta i databasen, sedan konverteras alla värden till string och tilldelas unika variabelnamn, t.ex. des för beskrivning och acc för kontonummer. En IF-sats jämför sedan dessa variabler mot de värden som finns insatta i de olika fälten, om alla värden är identiska går programmet vidare och stänger arbetsordern utan att vidta någon åtgärd. Om något eller flera värden inte är identiska tillkallas metoden FormSaveBox_Load (Kod 27).

Kod 18. Metod för jämförelse av värden

```
private void saveCheck()
{

   try
   {
        SqlCommand cmd = new SqlCommand("SET DATEFORMAT dmy "
        + "SELECT jo.description, jo.account, jl.client, jl.address , jl.printDate,

        ***
        while (check.Read())
        {
              datePrint = datePrinted.Value.ToShortDateString();
              string des = check[0].ToString(), acc = check[1].ToString(),

             ***

             if (textBoxType.Text == des && textBoxAccount.Text == acc && textBoxClient.Text
             ***

             {
                savePrint = "stangdirekt";
                saveState = "";
             }

             ***

 }
```

Med SQL-strängen UPDATE SET WHERE går det enkelt att uppdatera värden till databasen, dock måste det här kontrolleras vilka av datumfälten som faktiskt är ifyllda, annars kommer alla datumfält att adderas till SQL-strängen (Kod 19). Detta görs enklast med en IF ELSE-sats där man specificerar att om ett datumfält är ikryssat (Checked), annars läggs ett tomt värde (null) in. Fältet för uppskattad arbetstid

(numericEstimated) måste konverteras till string för att rätt antal timmar skall tillföras. Om uppdateringen lyckats meddelas användaren om detta med MessageBox.Show. (W3Schools, u.å.n.)

För att få rätt datumformat på de datum som hämtas från databasen måste de konverteras med:ToShortDateString(), vilket skulle motsvara SET DATEFORMAT DMY i en SQL -sträng. (Dotnetpearls, u.å.d.)

Kod 19. Uppdatering av data till databasen

```
public void Save()
{

    try
    {
          string time = numericEstimated.Value.ToString();
          StringBuilder updStr = new StringBuilder();
          SqlDataReader updaterl = null;


          updStr.Append("SET DATEFORMAT dmy "
                      + "UPDATE ee_jobOrder_list "
                      + "SET client = '" + textBoxClient.Text + "', "
                      ***
          if (dateStarted.Checked)
          {
             updStr.Append("startDate = '" + dateStarted.Value.ToShortDateString() + "',");
          }
          else
          {
              updStr.Append("startDate = null,");
            }
            ***

             updStr.Append("materialList = '" + textBoxMat_List.Text + "', "
             + "' WHERE jobNr = '" + comboBoxNr.Text + "', "

             ***

           if (savePrint == "save")
          {
               MessageBox.Show("Uppdateringen lyckades!");
          }
          ***
      }
 }
```

För att förhindra en situation där en användare glömt att spara ändringar i en arbetsorder tillkallas metoden FormOrder_FormClosing varje gång sidan stängs (Kod 20). Om variabeln savePrint är tom tillkallas metoden saveCheck som kontrollerar att den nuvarande informationen på arbetsordern motsvarar den som finns sparad i databasen. Om denna information inte är identisk (saveState = ”olika”) kommer en dialogruta att öppnas där användaren meddelas att denna har osparade ändringar i

arbetsordern och frågar om han /hon vill spara dessa, om användaren väljer ”Ja” tillkallas metoden Save() varefter fönstret stängs. Om valet är ”Nej” stängs fönstret utan att någonting sparas.

Kod 20. Exekveras då användaren stänger FormOrder

```
private void FormOrder_FormClosing(object sender, FormClosingEventArgs e)
{
    ***

    else
    {
        if (savePrint == "")
        {
            saveCheck();
        }

        try
        {
            if (saveState == "olika")
            {
                FormSaveBox formSaveBox = new FormSaveBox();
                formSaveBox.ShowDialog();

                if (formSaveBox.DialogResult == DialogResult.Yes)
                {
                    Save();
                    savePrint = "";
                }

                ***
            }

            ***
    }
}
```

3.4.15 Utskrift av befintliga arbetsorder

När knappen för utskrift klickas (Skriv ut) kommer återigen kontrollen för att alla värden i arbetsordern stämmer överens med de värden som finns databasen, saveCheck(), att exekveras (Kod 21). Orsaken till detta är att det inte skall förekomma utskrivna arbetsorder med värden som inte finns insatta i databasen. Om värdena inte stämmer överens tillkallas automatiskt Save() som sparar de aktuella värdena till databasen. Dock sker denna process i bakgrunden och användaren meddelas aldrig om detta, sedan tillkallas metoden print() som sköter själva utskriften av arbetsordern (se Kod 22).

Kod 21. Metoden för ”Skriv ut” knappen

```
private void pictureBoxPrint_Click(object sender, EventArgs e)
{
    savePrint = "print";
    saveCheck();
    if (saveState == "olika")
    {
        Save();
        print();
    }

    else
    {
        print();
    }
}
```

Då en arbetsorder skall skrivas ut tillkallas först metoden print() (Kod 22), här sparas en variabel ”s” som innehåller den nuvarande storleken av formuläret, varefter storleken justeras till ett förbestämt värde som passar till utskrift på ett A4 – pappersark. Bakgrundsfärgen på utskriften sätts till vit med ActiveForm.BackColor, this.Refresh() ritar upp bilden över formuläret på nytt för att undvika eventuella suddigheter i bilden. För att kunna fånga en bild av det nuvarande aktiva fönstret tillkallas CaptureScreen(). Efter detta återgår fönstret till den storlek som finns sparad i variabeln s. Här används även en förhandsgranskning av utskriften (printPreviewDialog1), om användaren godkänner utskriften skickas den vidare till skrivaren (printDocument1.Print()).

Kod 22. Metoden för utskrift av arbetsorder

```
private void print()
{

    Size s = this.Size;
    this.Size = new Size(1132, 685);
    FormOrder.ActiveForm.BackColor = Color.White;
    changeVisibilityDateBox(false);
    this.Refresh();
    CaptureScreen();

    ***
        if (printPreviewDialog1.ShowDialog() == DialogResult.OK)
        {
             printDocument1.Print();

        }
```

Metoden CaptureScreen() skapar en bild över den valda arbetsordern med hjälp av Graphics som senare skickas till skrivaren. Här sparas återigen storleken på fönstret som en variabel s med bredden (s.Width) samt höjden (s.Height). Själva bilden sparas i formatet bitmap (new Bitmap). (Microsoft, u.å.h.)

Kod 23. Metoden för att skapa en bild av arbetsordern

```
private void CaptureScreen()
{
    Graphics mygraphics = this.CreateGraphics();
    Size s = this.Size;
    memoryImage = new Bitmap(s.Width, s.Height, mygraphics);

    ***

}
```

Då den enda skillnaden mellan ett datum som är ifyllt jämfört med oifyllt är ett kryss framför själva datumfältet, kan missförstånd lätt uppstå om vilka datum som egentligen är ifyllda. Detta kan man lösa genom att gömma de oifyllda datumfälten från utskriften helt och hållet, i det här fallet med metoden changeVisibilityDateBox (Kod 24). Här tilldelas alla datumfält ett boolskt värde (showItem) beroende av om de är ikryssade eller inte, detta värde används sedan av metoden changeVisibilityOneDateBox (se Kod 25).

Kod 24. Metoden för hämtande av visibility

```
private void changeVisibilityDateBox(Boolean showItem)
{
    changeVisibilityOneDateBox(datePaused, showItem);
    changeVisibilityOneDateBox(dateStarted, showItem);
    changeVisibilityOneDateBox(dateResumed, showItem);
    ***
}
```

Metoden i Kod 25 visar hur datumfält som inte är ifyllda görs osynliga genom att använda det boolska värdet (showItem) från changeVisibilityDateBox. Om detta värde är ”true” betyder det att datumet är ifyllt och kommer att visas på utskriften, om värdet är ”false” görs datumfältet osynligt.

Kod 25. Metoden för ändrande av visibility

```
private void changeVisibilityOneDateBox(DateTimePicker dtp, Boolean showItem)
{
    if (showItem)
    {
        dtp.Visible = true;
    }
    else if(!dtp.Checked)
    {
        dtp.Visible = false;
    }
}
```

Då en arbetsorder är slutförd och användaren kryssat i slutdatum samt sparat arbetsordern kommer alla fält förutom datumfälten för ”Faktura skickad”, ”Betalning mottagen” och ”Inlämningsdatum” att vara låsta och kan alltså inte ändras mera.

Om användaren hör till admin -kategorin kan denne dock låsa upp dessa fält tillfälligt om någonting felaktigt lagts in i arbetsordern, detta sker genom att användaren klickar på en knapp som blir synlig då en arbetsorder är låst (Lås upp). Denna händelse tillkallar metoden pictureBoxReOpen som aktiverar alla fält (Enabled = true) men endast om användaren hör till admin. I annat fall dyker en textruta upp som meddelar användaren om att denne inte har rättigheter att utföra detta (Kod 26).

Kod 26. Metoden för att låsa upp en arbetsorder

```
private void pictureBoxReopen_Click(object sender, EventArgs e)
{
   if (admin)
   {
      textBoxClient.Enabled = true;
      textBoxAddress.Enabled = true;
      textBoxType.Enabled = true;
      ***
   }

   else

       MessageBox.Show("Du har inte rättigheter att utföra detta!");

 }
```

3.4.16 Ja/Nej -dialogfönster (FormSaveBox)

Även om man kan konstatera att gratisversionen av MS C# innehåller de flesta funktioner en programmerare behöver finns även vissa begränsningar, en av dessa är möjligheten att ändra texten i dialogrutor. Det är inte möjligt att använda sig av någon annan text på knappar än de befintliga (Yes/No/Cancel). När vi vill använda en dialogruta för påminnelse åt användaren om denne glömt att spara gjorda ändringar till databasen är det möjligt att definiera ett skilt formulär som agerar dialogruta och här sätta in valfri text på knapparna, i detta fall Ja och Nej.

Då en användare försöker stänga FormOrder och glömt att spara någon ändring på den aktuella arbetsordern dyker en dialogruta upp som meddelar om detta och frågar om denne vill spara dessa ändringar (se Figur 6).

För att åstadkomma detta anger vi först under metoden FormSaveBox_Load() vilka knappar som gör vad (Kod 27). Här deklareras två stycken knappar (buttonYes och buttonNo), dessa förknippas sedan med DialogResult.Yes för Ja- knappen resp. DialogResult.Cancel för Nej -knappen samt this.Close() som stänger detta formulär. SuspendLayout() och ResumeLayout() är till för att förhindra konflikter som kan uppstå då knapparna definieras. (Codeguru, u.å.c.)

Kod 27. Metoden för definierande av knappar

```
public void FormSaveBox_Load(object sender, EventArgs e)
{
    this.buttonYes = new System.Windows.Forms.Button();
    this.buttonNo = new System.Windows.Forms.Button();
    this.SuspendLayout();
    buttonYes.DialogResult = System.Windows.Forms.DialogResult.Yes;
    buttonNo.DialogResult = System.Windows.Forms.DialogResult.Cancel;
    this.ResumeLayout(false);

}
```

Om användaren klickar på ”Ja” -knappen tillkallas metoden buttonYes_Click som returnerar DialogResult.Yes varefter dialogrutan stängs (Kod 28).

Kod 28. Metoden för ”Ja” knappen

```
public void buttonYes_Click(object sender, EventArgs e)
{
    this.DialogResult = DialogResult.Yes;
    this.Close();

}
```

Metoden buttonNo_Click fungerar på exakt samma sätt förutom att DialogResult.Cancel returneras istället (Kod 29).

Kod 29. Metoden för "Nej" knappen

```
private void buttonNo_Click(object sender, EventArgs e)
{
    this.DialogResult = DialogResult.Cancel;
    this.Close();

}
```

4 Testning

En testperiod på tre månader för programmet inleddes i mitten av oktober 2010, syftet var att fram till årsskiftet hitta och lösa eventuella buggar eller andra problem samt att ge alla användare en möjlighet att komma med olika åsikter om funktionaliteten och förbättringsförslag. Den slutgiltiga versionen av programmet togs i bruk i slutet av januari 2011.

4.1 Buggar

Då ett nytt program testas dyker det oftast upp ett antal buggar, dvs något fel i programmet (oftast i programkoden) som t.ex. gör att det slutar fungera eller kraschar. Förvånandsvärt nog hittades under denna tid endast en bugg i programmet. Då användaren i ett textfält tryckte på enter för att byta rad och sedan sparade arbetsordern visades denna nya rad som ett märkligt tecken i DGV. Lösningen var dock väldigt enkel, nämligen att aktivera multiline (multipla rader) för de textfält som innehöll text på flera rader.

4.2 Användar -feedback

I stort sett var personalen nöjd med programmet men ett antal saker fattades eller var felaktiga. Bland dessa var beräkning och visning av totala antalet timmar för samtliga montörer i rutan för "Deltagit i arbete" och en ny datumväljare då arbetsordern tagits emot av personalen vid materiallagret. Även vissa textfält visade sig vara för små och behövde förstoras, främst fältet för beskrivning. Dessutom ville

användarna att resultaten i DGV skulle uppdateras även då text manuellt raderas från något textfält. Detta gjordes så att användaren trycker på enter-tangenten då denne vill uppdatera sökresultaten. Ett missförstånd hade även skett för de två datumväljare som finns i huvudfönstret, meningen var att de skulle visa sökresultaten för hela intervallet mellan två datum och inte endast dessa två datum. I övrigt tyckte användarna att användargränsnittet var tydligt och lätt att förstå.

5 Slutsatser

När detta projekt startade mot slutet av mars månad 2010 fanns det två olika ideér för tillvägagångssätt att utveckla systemet. Det första alternativet var att utveckla ett webbgränssnitt med hjälp av webdesignverktyget Joomla!, tanken bakom detta var att få ett system som är tillgängligt för personalen oavsett om de befinner sig på kontoret eller ute på fältet. Den andra möjligheten var att bygga upp ett program med hjälp av något programmeringsspråk, t.ex. C# (detta var också den linje vi senare valde att följa). Min handledare presenterade en skiss över ett flödesschema som exempel på hur programmet kunde byggas upp. Det första alternativet slopades då vi kom fram till att utvecklingen av ett sådant system troligen skulle vara väldigt tidskrävande med tanke på att varken jag själv eller min handledare vid EE hade någon större erfarenhet av Joomla!.

I början var meningen att en testversion av programmet skulle tas ibruk redan under sommaren 2010, men detta kom senare att skjutas upp till hösten eftersom det inte skulle ha varit någon större idé med att ta i bruk ett program för testning då de flesta av personalen befinner sig på semester. Det skulle troligtvis inte heller varit möjligt för min egen del att få fram en brukbar version av programmet så tidigt med tanke på den ringa erfarenhet jag hade av programmeringsprojekt och det faktum att jag arbetade ensam.

Innan jag började jobba med detta arbete hade jag ingen erfarenhet överhuvudtaget av just C#, det enda jag hört ryktesvis var att C# till en del liknar C++ som jag hade lite erfarenhet av från tidigare. När jag väl hade kommit igång med själva kodandet blev det tydligt att det nog fanns en hel del likheter med just C++, och då det uppstod

situationer där jag inte riktigt visste hur ett specifikt problem skulle lösas var både olika forum och artiklar på internet samt min handledare vid EE till stor hjälp.

Mot slutet av januari 2011 togs den senaste versionen av programmet i bruk och jag avsade mig ansvaret för vidareutveckling. Både jag och arbetsledningen var nöjda med slutresultatet av projektet då de mål som sattes upp under våren förra året nu hade uppnåtts.

Källförteckning

ASPNet, u.å.a. Call one event into another event using c#.
http://forums.asp.net/t/1114513.aspx.
(hämtat: 12.08.2010)

C# Programming language, u.å.b. Wikipedia.
http://en.wikipedia.org/wiki/C_Sharp_(programming_language).
(hämtat: 20.02.2011)

Codeguru, u.å.c. Creating a Custom Message Box.
http://www.codeguru.com/csharp/.net/net_general/eventsanddelegates/article.php/c13193.
(hämtat: 17.12.2010)

Dotnetpearls, u.å.d. C# DateTime Format Examples.
http://dotnetperls.com/datetime-format.
(hämtat: 14.08.2010)

Dotnetpearls, u.å.e. C# DateTime.Parse SQL Dates.
http://dotnetperls.com/datetime-sql.
(hämtat: 14.08.2010)

Dotnetpearls, u.å.f. C# string.IsNullOrEmpty Method.
http://dotnetperls.com/isnullorempty.
(hämtat: 24.08.2010)

Eggheadcafe, u.å.g. How to get the all cell values from DataGridView in Button Click event. http://www.eggheadcafe.com/community/aspnet/2/10044280/how-to-get-the-all-cell-values-from-datagridview-in-button-click-event.aspx.
(hämtat: 12.08.2010)

Microsoft, u.å.h. Code: Printing the Form (Visual C#).
http://msdn.microsoft.com/en-us/library/aa287529%28v=vs.71%29.aspx.
(hämtat: 02.06.2010)

Microsoft, u.å.i. How to: Bind Data to the Windows Forms DataGridView Control. http://msdn.microsoft.com/en-us/library/fbk67b6z.aspx.
(hämtat: 23.05.2010)

Microsoft,u.å.j. How to: Get a Value from Another Form. http://msdn.microsoft.com/en-us/library/f6525896(VS.90).aspx.
(hämtat: 12.08.2010)

Microsoft, u.å.k. Populate ComboBox from SQL with and without using Stored Procedure. http://social.msdn.microsoft.com/Forums/en-US/Offtopic/thread/24a391d2-8838-4d44-993b-a43046a42b05.
(hämtat: 12.05.2010)

Microsoft SQL Server, u.å.l. Wikipedia. http://en.wikipedia.org/wiki/Microsoft_SQL_Server.
(hämtat: 20.02.2011)

Transact-SQL, u.å.m. Wikipedia. http://en.wikipedia.org/wiki/Transact-SQL
(hämtat: 20.02.2011)

W3Schools, u.å.n. SQL UPDATE Statement. http://www.w3schools.com/sql/sql_update.asp.
(hämtat: 11.07.2010)